# Towards Ubiquitous Intelligent Computing: Heterogeneous Distributed Deep Neural Networks

Zongpu Zhang, Tao Song, *Member, IEEE,* Liwei Lin, Yang Hua, Xufeng He, Zhengui Xue, Ruhui Ma, *Member, IEEE,* and Haibing Guan, *Member, IEEE*

**Abstract**—For the pursuit of ubiquitous computing, distributed computing systems containing the cloud, edge devices, and Internet-of-Things devices are highly demanded. However, existing distributed frameworks do not tailor for the fast development of Deep Neural Network (DNN), which is the key technique behind many intelligent applications nowadays. Based on prior exploration on distributed deep neural networks (DDNN), we propose Heterogeneous Distributed Deep Neural Network (HDDNN) over the distributed hierarchy, targeting at ubiquitous intelligent computing. While being able to support basic functionalities of DNNs, our framework is optimized for various types of heterogeneity, including heterogeneous computing nodes, heterogeneous neural networks, and heterogeneous system tasks. Besides, our framework features parallel computing, privacy protection and robustness, with other consideration for the combination of heterogeneous distributed system and DNN. Extensive experiments demonstrate that our framework is capable of utilizing hierarchical distributed system better for DNN and tailoring DNN for real-world distributed system properly, which is with low response time, high performance, and better user experience.

**Index Terms**—heterogeneous distributed deep neural network, HDDNN, deep neural network, DNN, Internet of Things, edge computing, cloud computing

## 1 Introduction

In recent years, thanks to the development of machine learning, many intelligent applications have changed people's life unconsciously, e.g., voice assistant and biometric authentication. With the rapid advancement in Convolutional Neural Networks (CNN) and Deep Neural Networks (DNN), learning based artificial intelligence algorithms have become the mainstream and have revolutionized many research fields, for example, digital image processing [1], [2], image classification [3], [4], speech translation [5], [6], speech recognition [7], [8] and so on. Currently, the most common solution to the well-known great need of computational resources while using DNNs is to rely on powerful computing units, e.g. GPU, or the cloud service. Despite much progress on adapting DNNs to mobile devices, limited progress has been made on exploring the collaboration of the mobile and cloud computational power.

Ubiquitous computing involves spreading, or distributing the computational resources across multiple devices which might be located in different places. One of the solutions involves utilizing the hierarchical distributed system containing the cloud, edge (fog), and end devices. In such system, computation intensive services are packed and processed mainly by the cloud. Meanwhile, edge and end devices share and distribute the service, making ubiquitous computing powerful, but unperceivable to end users (like the electricity in daily lives). In recent years, the hierarchical scalable distributed computing system has made a fast progress [9], [10], [11], which shows its importance. Along with the development of Internet technologies, computing nodes located in different places geographically are able to form a cooperative system providing ubiquitous computing services for users. Large number of end devices connected together, forming an Internet of Things (IoT), has become another hot topic in IT industry recently. With advantages such as close to the data provider, sensors, distributed system utilizing IoT devices is expected to be the next breakthrough of ubiquitous computing for the daily applications. As a type of well-organized framework, distributed systems are generally expected to have other features, e.g., high availability, high scalability and robustness over large scale and heterogeneous devices, which is different from DNNs. Such computing system has not yet adapted to the fast progress of DNNs, for instance, the workload of a DNN-based service is commonly treated the same as traditional computational intensive cloud services.

In addition, current DNNs focus more on improving the performance based on single device deployment, such as MobileNet [12] and SqueezeNet [13]. Limited progress has been observed on designing and reasoning DNNs for the hierarchical distributed system. As an early attempt, Distributed Deep Neural Network (DDNN) [11] is proposed as a DNN design pattern for better utilization of the distributed system hierarchy. Despite the great novelty, certain limitations are observed, including: (a). they more focus on the power eating training procedure on end devices, (b).

- *Z. Zhang, T. Song, L. Lin, X. He, Z. Xue, R. Ma and H. Guan are with the Shanghai Key Laboratory of Scalable Computing and Systems, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China, 200240.*
  *E-mail: {zhangz-z-p, songt333, llw02_02, hexufeng, zhenguixue, ruhuima, hbguan} @sjtu.edu.cn*
- *Y. Hua is with EEECS/ECIT, Queen's University Belfast, UK, BT3 9DT*
  *E-mail: Y.Hua@qub.ac.uk*

*Manuscript received xx, xx; revised xx, xx.*

they use DNNs with exactly the same structure on all kinds of devices, and (c). they only validate their framework with limited task-specific experiments. More sophisticated design of DNNs should be made to fit the real-world distributed computing system.

To this end, we propose a novel Heterogeneous Distributed Deep Neural Networks (HDDNN) framework, trying to bridge the DNN and the hierarchical distributed system by mutually optimization in two research fields. On the one hand, we improve distribution computing framework in order to better support DNNs, including (a). distributed computing node heterogeneity for better resource utilization, (b). fully supported parallel computing with a mapping based scheduling algorithm, and (c). privacy protection routine and robustness. On the other hand, we also fit DNNs to the real-world distributed computing systems. Specifically, we design (a). distributed neural network heterogeneity for better system performance, and (b). system task heterogeneity for better user experience. The contributions of this paper include:

1) We propose a novel Heterogeneous Distributed Deep Neural Networks (HDDNN) over the cloud, edge (fog) and end devices.
2) We improve distributed computing framework to better support DNNs.
3) We build distributed-computing-systems-aware DNNs in order to fully utilize ubiquitous computing resources.
4) We conduct intensive evaluations to demonstrate the advantages of our proposed framework, including short response time, high accuracy, better hardware usage, high scalability, privacy protection, and fault tolerance.

The source code and pre-trained models are available at: https://github.com/Maphist0/HDDNN.

The remaining parts of this paper are organized as follows. We review recent researches related to distributed computing hierarchy and neural networks in §2. We introduce our proposed framework in §3. The evaluation of our simulation framework is illustrated in §4. Finally, §5 is our conclusion.

## 2 Related Works

In this section, we review recent research related to distributed computing hierarchy in §2.1, application of deep neural networks to computer vision problems in §2.2, and the framework of distributed deep neural networks in §2.3.

### 2.1 Distributed Computing Hierarchy

Cloud computing has long been one of the most popular research directions in the IT industry since 2006 [14]. Its scalable infrastructure enables companies to run their business models on various cloud computing platforms, for instance, Amazon AWS [15], and Microsoft Azure [16]. With other functionalities offered by clouding computing, e.g. resource virtualization [17], software as a service (SaaS) [18], has been widely used in our daily life. However, real-world problems like high latency and sensitive data protection [19], [20] may degrade the user experience, and they are great challenges facing cloud computing researchers.

Driven by both the advancement of 5G communications and network technologies, a paradigm shift in migrating cloud computation to network edges has been seen [21]. The trend that more and more data are generated at the network edge pushes the development of edge computing [22], which involves processing data directly at edge devices. Due to ongoing differences on the concept of edge and fog, we use edge mainly in the following discussion and do not explicitly distinguish them. Previous works [23], [24] have been introduced to exploit edge computing. They show the possibility to eliminate the drawback of long propagation delays compared with cloud computing in real-world applications. Other benefits are also provided by edge computing, such as lower response time [25] and lower energy consumption [26], using typical end devices like a gateway, a micro data center, and the cloudlet [24].

With the rapidly increasing number of smart devices deployed around us , the next wave in the era of computing is predicted to be the Internet of Things (IoT) [19], [27], [28]. With the development of Internet technologies, sensor manufacturing, and Near Field Communication technologies, a significant boost in the number of sensors is observed in recent years [29], [30]. One of the consequences of such phenomenon lies in the fact that these sensors generate huge amount of data, which can not be handled by cloud computing alone. With the sensor network, which is a key component of IoT, data can be quickly processed first on local end devices [27]. The connection of IoT devices can support many applications, including Healthcare system [31], [32], [33], Smart city / Smart home [34], [35], [36], Video surveillance [37], [38], [39] and so on. In terms of mobile devices, though some researchers categorize them into edge devices due to its computation power [40], [41], [42], we consider their energy limitation and treat them as end devices in the following discussion.

The complexity of real-world applications and systems grows tremendously in recent years, showing high demand of joint ubiquitous computing models by the industry. With that said, three layers of computational models, i.e. Cloud, Edge, and End, forms a new hierarchical structure. Such hierarchical structures are developed to meet various requirements, e.g., fast response, low latency, high availability, reliability, manageability, and low cost [9].

### 2.2 DNN for Computer Vision

As one of the most popular subjects in the computer science society, computer vision problems such as image classification have long been a test bed for DNNs. Many types of DNN architecture have been proposed to meet different real-world requirements, such as high performance, high speed, and low utilization of computational and memory resources. A milestone for the application of neural networks in computer vision problems was the introduction of AlexNet [43], which considerably outperformed traditional methods in the ImageNet Large Scale Visual Recognition Challenge [44] in 2012. Later, CNNs with smaller convolutional filters and deeper network architecture were proposed, showing outstanding performance on many challenges, e.g., VGG Net [45]. Residual Network [3], and

its successor Wide Residual Network (WRN) [46] ease the difficulty of training much deeper DNNs by introducing residual functions to solve the gradient vanishing problem. However, typical high-performance DNNs can only work with powerful hardware and a great amount of memory resources, which is unrealistic for low power consumption, portable and low-cost IoT devices or mobile devices.

Many attempts have been made to fit DNNs on mobile devices [12], [47], [48], from the computation or the storage aspect. Binarized neural network (BNN) [47] simplifies weights in linear and convolutional layers to -1 and 1, for less memory and computation. MobileNet [12] uses streamlined architecture with depthwise and pointwise convolutional layers for fewer parameters and higher speed. Extensive experiments for various tasks on several datasets, e.g., MNIST [49] and CIFAR [50], show that they can achieve acceptable performance with much less computation and smaller model size compared with state-of-the-art structures.

### 2.3 Distributed DNN

Related work of applying large-scale DNNs on distributed systems traditionally focuses on speeding up the training phase. Some frameworks have proved the efficiency of distributing neural networks on large-scale computer clusters with the help of both data and model parallelism. DistBelief framework [51] employs an asynchronous stochastic gradient descent procedure, Downpour SGD and a distributed batch optimization framework, Sandblaster. It has been shown in [51] that the system can train up to 30x larger networks for up to 12x speedup (on 81 machines) on a CPU based computer cluster with state-of-the-art performance on ImageNet dataset [43]. On the other hand, FireCaffe [52] utilizes reduction tree based synchronous stochastic gradient descent algorithm and at the same time implements the framework on a large scale graphics processing unit (GPU) cluster. It achieves up to 47x speedup (on 128 GPUs) for a DNN with state-of-the-art performance on ImageNet. Different optimization algorithms, specifically different SGD algorithms (e.g., Gossiping SGD [53]) have also been proposed for better performance and lower training time of DNNs on distributed systems.

However, much less research has been done for distributing DNNs on a hierarchical distributed system, where the hardware, location and network resources are dramatically different for each type of computing nodes. Researchers traditionally focus less on edge and end devices due to the limitation of computational and memory resources. With the development of IoT devices and mobile devices, it is now possible to implement neural network algorithms on such end devices.

In [11], a distributed framework was proposed to apply DNNs on the cloud, the edge (fog) and end devices, where the DNNs are partitioned and the shallow part is put on less powerful computing nodes (edge/end devices) with an algorithm determining the local exit point. The system can scale both horizontally (for neural network size) and vertically (for geographical span) with features, such as fault tolerance and privacy protection. By testing on a multi-view camera image classification dataset, it is shown that the system can achieve high accuracy and at the same time reduce the communication cost by over 20x.

## 3 The Framework of HDDNN

As illustrated in Figure 1, our proposed framework consists of three levels of heterogeneity: (a). distributed computing node heterogeneity, (b). computing node neural network heterogeneity, and (c). system task heterogeneity. Based on the observation of real-world situations and the research on related work, the following assumptions are made in our discussion.

1) We focus on the testing phase of neural network algorithms, rather than the training phase.
2) All tasks are assigned from end devices, in a bottom-to-top way. For simplicity, we only consider the case where one end device assigns tasks.
3) End devices cannot process for a long time due to power limit. Edge devices are more powerful than end devices, and the power limit is ignored on them. The cloud doesn't have limitation on computational resources and power consumption.
4) To protect privacy, all end devices are considered dependable, edge devices are considered partly dependable and partly undependable. The cloud is regarded as undependable.

Considering that nodes far away from end users might not be more powerful than nodes close to end users in real-world situations, we mainly separate cloud, edge and end devices by their distances with end users in our proposed system for the sake of generality. End devices stand for those located relatively close to the end user. For example, mobile phones owned by users, IoT devices inside a smart home, and smart wearable devices on the end user are typical end devices. Since end devices tend to suffer from low performance, low throughput, and high device occupancy, it is not a preferred methodology to simply process data at end devices. Thus, it is more efficient to upload the data that need to be processed or have been pre-processed by end devices (e.g., encryption) to more powerful computing nodes for better results. The cloud represents devices that are far away from the end user, which is generally considered very powerful. A cluster of machines from a service provider, or some powerful workstations in some companies can be treated as cloud for their longer distance to end users. Devices stay between end devices and the cloud are called edge devices, such as routers and small data centers located between users and the service provider.

### 3.1 Distributed Computing Node Heterogeneity

Our framework is built upon hierarchical distributed systems, including cloud, edge devices, and end devices. Existing ways for deep learning based service providers to handle data from end users involves uploading all data to their processing node, e.g. a cloud. However, such framework has many disadvantages.

1) Long response delays are likely to be generated, due to unpredictable network connection quality between end users and the service provider.

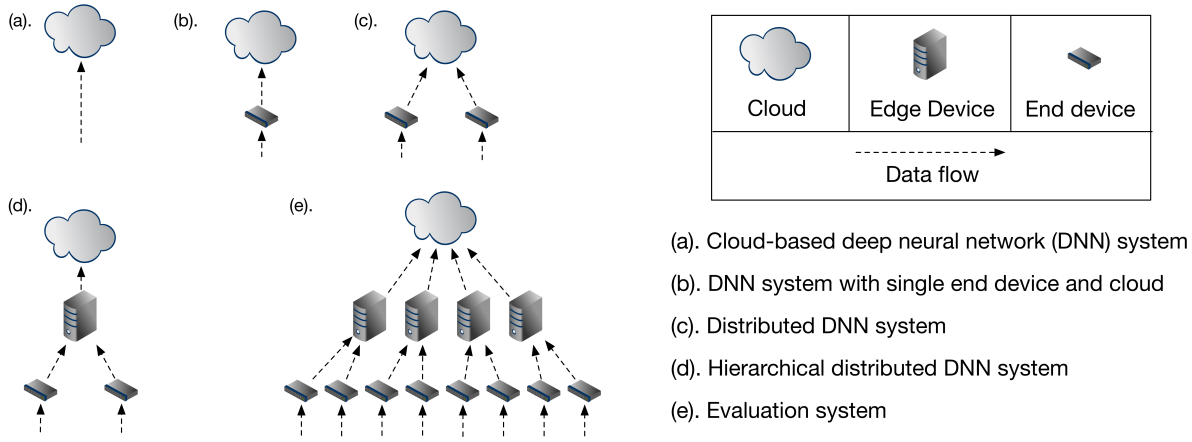


Fig. 1. Illustration of distributed deep neural network system over the cloud, edge devices and end devices. Traditionally, users have to upload all data to the cloud for deep learning applications (part a). With the advancement in mobile computing, user's data may first be pre-processed on one (or multiple) end device(s) located close to the user for lower response time (part b and c). Hierarchical distributed DNN system (part d) utilizes edge devices to bridge the computational power, network latency, etc. between end devices and the cloud. We evaluate our proposed HDDNN framework on a system with 8 end devices, 4 edge devices, and 1 cloud (part e).

2) Large amount of data are generated every second with the development of IoT devices, which is likely to cause unacceptable long communication latency by conventional computing, e.g. uploading all data to the cloud.
3) The computational resources on other end devices around the end user, and edge devices between the end user and the cloud are wasted.
4) The privacy issue is likely to happen when uploading sensitive data to the cloud.

Instead of relying on a cloud remotely, our framework distributes the computing on different types of computing nodes. Since different types of nodes have their own features, our framework tries to meet their requirements and makes the best use of them. End devices are closer to users, but they have many hardware restrictions, such as the limitation in memory and power consumption. We set relatively easier tasks for end devices to respond quickly. Edge devices then are set for moderately difficult tasks to compensate the low performance of end devices in slightly longer time. Cloud is set for the most difficult tasks, hoping to make full use of its powerful hardware and provide the highest performance.

Besides hardware, distributing computing node heterogeneity also requires considering the network latency for different types of devices. Based on the assumption that all tasks are generated from end devices, the hierarchy of end devices, edge devices, and the cloud naturally represents the network latency for them. Other end devices located in the same network, or close to the task assigner have the lowest network latency. Edge devices then have moderate network latency, and the cloud has the highest latency. By setting tasks for end devices and edge devices, our framework can be adaptable to different network configurations and efficiently process the data.

### 3.2 Neural Network Heterogeneity

Instead of loading neural networks with the same structure on all types of devices in the distributed system hierarchy, as in DDNN [11], devices in our proposed framework are loaded with different neural networks. It is straightforward that loading a binary network on the cloud like DDNN is a waste of computing resources. On the other hand, loading a typical DNN with several hundreds of layers on end devices is unrealistic. The goal of enabling neural network heterogeneity is to fully utilize the features of each device, including memory limitation, computational capability, and network status, etc.

For three types of devices (i.e., cloud, edge device, and end device), different factors should be emphasized when we choose the neural networks for it. The cloud has the most powerful hardware in the distributed hierarchy. Therefore, the primary metric for choosing neural networks on the cloud is the performance, e.g., the lowest error rate for image classification. Edge devices have the medium computational capability meanwhile they are significantly closer to end users. DNNs which balance between the performance and time delay (including processing time on edge devices, and communication overhead) are chosen for them. End devices have the most restricted condition, where choices of neural networks could be limited by memory size and computational capability. A shallow or more optimized neural network should be chosen for end devices. Factors including less response time, low power consumption, and low memory usage should be considered ahead of the performance of neural networks.

As discussed in §3.1, tasks issued from end users are first processed in computing nodes which stay close to the end users. With a fast neural network loaded on the end devices, the response time is dramatically reduced compared with uploading the task to the cloud. It is understandable that the performance of the first response may not be ideal due to the shallow model on the end devices.

### 3.3 System Task Heterogeneity

We propose a heterogeneous system task for the consideration of real-world situations. Sometimes a coarse-grained result is enough for the user to draw a conclusion, or react correspondingly, e.g. knowing certain object as one kind of 'fruit' may be enough instead of identifying the exact category, e.g., 'apple'. With system task heterogeneity, neural networks that are loaded on devices with fewer computational resources can be modified to output a coarse-grained response. By doing so, instead of forcing the neural network to output a more meaningful but computationally expensive fine-grained response, the calculation time of such devices can be improved to a usable level. Combined with distributed computing node heterogeneity (§3.1) and neural network heterogeneity (§3.2), our framework is designed to load different neural networks for different tasks on different types of machines in the distributed hierarchy. Relatively shallow networks for coarse-grained tasks are loaded on end devices for fast response time, low memory requirement and low power consumption. DNNs for fine-grained tasks are loaded on edge devices, which produce much better results than the end devices. Even stronger DNNs are loaded on the cloud for the highest possible performance of fine-grained tasks. Resources on each type of computing nodes are fully utilized and optimized, and the overall system outperforms the individual device in different ways, such as the response time and the accuracy.

Comparing side-by-side to DDNN, besides the aforementioned advantages, system task heterogeneity enables better resource usage, better user experience and more flexible system design approach for a heterogeneous distributed computing system. We present an example of applying system task heterogeneity to the DDNN framework. Instead of performing 3-class classification tasks (person/bus/car) in both local exit and cloud exit, end devices may perform the classification tasks like before and output via the local exit, while the cloud can perform anomaly detection using the pre-processed data from end devices to detect abnormal behavior of objects and output via the cloud exit. It is highly flexible while designing the task for various types of devices, so that each device's features such as the performance, power limit, and network situation get fully exploited.

### 3.4 Scheduling Algorithm and Scalability

The scheduling algorithm is important for optimizing the performance of the proposed system with data intensive jobs. To formulate and compare several types of scheduling algorithms, we present four schemes representative for different real-world situations. Following the assumptions at the beginning of §3, tasks to be processed in our framework are first sent to weaker nodes for encryption, then to stronger nodes for better results. For simplicity, suppose each task is processed on no more than two types of devices in the distributed hierarchy, and the first destination is an end device. Also assume that the speed of the job is determined by the scheduling scheme, processing time, and communication time. Other factors, e.g., the speed of assigning tasks to other devices, and the encryption time are not taken into consideration in this section.

As a baseline, **'End' scheme** involves sending tasks to all end devices in a sequential order. For example, iterate from end device 1, to end device 2, all the way to end device $N_{end}$, then back to end device 1. In ideal cases, the time to complete this job is expressed in eq. 1.

$$T_{End} = \frac{N_t}{N_{end}} \times (t^c_{end} + t^p_{end}) \tag{1}$$

where $T_{End}$ is the total time required in the 'End' scheme, $N_t$ is the total number of tasks in the job, $N_{end}$ is the number of end devices, $t^c_{end}$ and $t^p_{end}$ are the average **c**ommunication time (between two end devices) and **p**rocessing time of each task on end devices, respectively; The sum of $t^c_{end}$ and $t^p_{end}$ represents the average time of each task on an end device; The product of the average number of tasks received by each end device, $N_t/N_{end}$, and the time for each task forms the total required time of the 'End' scheme. Since end devices are generally close to the user, the communication time, $t^c_{end}$ for end devices is considerably low, which means the time to finish the job, $T_{End}$, is mainly controlled by the number of end devices $N_{end}$ and the processing time on end devices $t^p_{end}$, i.e. more end devices and shorter processing time contribute to lower overall complete time. However, this 'End' scheme has the worst overall precision because only shallow neural networks on end devices are used for evaluation. The overall precision $P_{End}$ can be treated as the average precision of neural networks on end devices ($p_{end}$).

$$P_{End} = p_{end} \tag{2}$$

**'End-Cloud' scheme** is an extension of the 'End' scheme, where all tasks are first processed on end devices, then sent to the cloud for further processing. In ideal cases, tasks for the cloud are assigned also in a sequential order, thus the time to complete this job is expressed as the combination of total time on end devices and the cloud.

$$T_{End-Cloud} = T_{End} + \frac{N_t}{N_{cloud}} \times (t^c_{cloud} + t^p_{cloud}) \tag{3}$$

$T_{end}$ follows eq. 1, $N_{cloud}$ for the number of cloud nodes, $t^c_{cloud}$ and $t^p_{cloud}$ for average communication time (between an end device and a cloud) and processing time of each task on cloud respectively. Since $t^c_{cloud}$ is generally considered large and unstable, the communication overhead then becomes the bottleneck of the 'End-Cloud' scheme in practice, instead of $T_{End}$ or $t^p_{cloud}$ (normally given the constant $N_{cloud}$). The overall precision $P_{End-Cloud}$ for this scheme is considered the best, mainly depending on the average precision of neural networks on the cloud ($p_{cloud}$).

$$P_{End-Cloud} = max\{p_{end},\ p_{cloud}\} = p_{cloud} \tag{4}$$

**'End-Edge' scheme** is a trade-off between processing time and performance. In ideal cases, $T_{End}$ and the overall time spent on edge devices determine the total finish time, and the average precision on edge devices determines the overall precision.

$$T_{End-Edge} = T_{End} + \frac{N_t}{N_{edge}} \times (t^c_{edge} + t^p_{edge}) \tag{5}$$

$$P_{End-Edge} = p_{edge} \tag{6}$$

With moderate network latency, the communication overhead is generally no longer the bottleneck of the entire

system. Also, since memory and computational resources on edge devices are considered much better than end devices, it is both affordable and effective to adopt relatively deeper neural networks on edge devices to balance the processing time and performance.

However, the above three schemes cannot fully utilize the distributed hierarchy, because at most two types of devices are used while processing the job. It is worth noting that devices in a higher hierarchy (e.g., edge devices compared to end devices) generally require longer time for each task due to higher network latency and longer processing time . Thus we propose a novel **'Mapping' scheme** as a scheduling method in our framework. It considers the ratio of the total processing time between end devices and edge devices, and assign the upper-level processing unit for each end devices instead of assigning sequentially. Intuitively, the number of end devices mapped to one edge device can be calculated by the ratio of the average time for each task on two types of devices.

$$N_{map} = \lceil \frac{t^c_{edge} + t^p_{edge}}{t^c_{end} + t^p_{end}} \rceil \tag{7}$$

It means that each subset of $N_{map}$ end devices sends the workload to one edge device, so that the total time for $N_{map}$ end devices to process the next batch of input data is no less than the time for one edge device to process. When all edge devices have been assigned to a certain number of end devices, the rest end devices are applied the same algorithm and mapped to the cloud. Tasks for end devices are still scheduled in sequential order for the highest usage of computational resources on end devices. The ratio of tasks sent to the cloud is calculated by the number of remaining end devices divided by the total number of end devices.

$$\theta_{cloud} = \frac{N_{end} - N_{edge} \times N_{map}}{N_{end}} \tag{8}$$

The overall performance is the weighted sum of the performance from edge devices and the cloud. The processing time for 'Mapping' scheme $T_{Mapping}$ is also a weighted sum of the total time for 'End-Cloud' scheme and 'End-Edge' scheme, which stays between $T_{End-Cloud}$ and $T_{End-Edge}$.

$$\theta_{edge} = 1 - \theta_{cloud} \tag{9}$$

$$P_{Mapping} = \theta_{cloud} \times p_{cloud} + \theta_{edge} \times p_{edge} \tag{10}$$

$$T_{Mapping} = \theta_{cloud} \times T_{End-Cloud} + \theta_{edge} \times T_{End-Edge} \tag{11}$$

Two advantages of applying the 'Mapping' scheme are straightforward: (a). by offloading certain ratio of tasks to the cloud, the precision can be improved from the 'End-Edge' scheme with controllable processing time overhead from the 'End-Cloud' scheme, and (b). higher utilization of devices in the distributed hierarchy. Since the main problem for the 'End-Cloud' scheme in practice is the bottleneck of high network delay in the cloud side, the 'Mapping' scheme reduces such effect by assigning part of the job to cloud, and the rest to edge devices for lower response time. On the other hand, those tasks that are assigned to cloud enjoy higher performance than that on the edge devices. This makes the scheduling of specific tasks according to their importance possible. We will discuss it in this paper. The scalability of the 'End' scheme and the 'End-Edge' scheme is theoretically better than the 'End-Cloud' due to the communication bottleneck of the 'End-Cloud'. However, we observe in practice that other factors, e.g., the speed of assigning tasks, become the bottleneck of the 'End' scheme, because the total time for the 'End' is relatively small. The scalability of 'Mapping' is more complex: (a). it is closer to 'End-Edge' when the number of end devices cannot satisfy the requirement of edge devices, but (b). it is closer to the 'End-Cloud' when the number of end devices are too large.

### 3.5 Privacy Protection

Privacy issue could happen when sensitive data are transferred to untrusted devices, or to a cloud. For example, hackers may intercept the transmitted data while being transferred to another device through an open network, or attack the cloud service provider to steal users' private data. Our framework prevents such cases from happening by encrypting sensitive data with a single layer neural network stripped from the neural network in a higher device hierarchy. The first convolutional layer of the neural network in the cloud (or edge) is downloaded on the end device as a light-weight encryption module. Instead of sending the source data, the output data from the encryption module, which is a multi-dimensional matrix containing floating point numbers, is sent to untrusted devices if necessary.

We show the privacy is protected by introducing the calculation of different typical layers in a CNN. The convolutional layer output $y$ for a $N \times N$ input $x$, with one $M \times M$ filter $\omega$, and without padding and strides has the form of eq. 12.

$$y^l_{i,j} = \sum_{a=0}^{M-1} \sum_{b=0}^{M-1} \omega_{a,b} \times x^l_{(i+a),(j+b)} \tag{12}$$

$l$ denotes the current depth of model. Calculating reversely from an output $y$ to the input image $x$ involves finding the inverse function $F^{-1}$ of the above convolution operation.

$$\hat{x}^l_{i,j} = F^{-1}(y^l_{i,j}) \tag{13}$$

It is clearly impossible to find the inverse function $F^{-1}$ without knowing the exact weight in all filters. Even if all $N_f$ filters are obtained by the attacker, calculating the original image data involves solving a $M \times M$ linear equation for $N^2$ times, which is unrealistic in practice.

One of the concerns of using convolutional layers to encrypt the source data is the computational overhead. Using the encryption module on end devices, firstly, the time to feed-forward the encryption neural network is taken into consideration for each task. Denote the time of using the encryption module from different devices as $t^e$, for instance, $t^e_{edge}$ for using the encryption module from the edge device, and $t^e_{cloud}$ from the cloud. Notice $t^e_{end} = 0$ due to the assumption that all end devices are dependable, i.e. no encryption is needed between two end devices. Secondly, since the amount of data transferred between two devices have changed from the raw image to the encrypted data, we denote the new transferring time as $\hat{t}^c$ (the subscript for the device type is hidden for a clearer look). Given a specific task, the processing time without encryption, $T_{raw}$, on any

type of devices is the sum of the communication latency and the processing time, as is discussed in §3.4. The time with encryption, $T_{encrypt}$, is the sum of the new communication latency, the processing time, and the encryption time. Thus the processing time overhead, $\Delta T$, introduced by the encryption is the difference of $T_{encrypt}$ and $T_{raw}$.

$$T_{raw} = t^c + t^p \tag{14}$$

$$T_{encrypt} = \hat{t}^c + t^p + t^e \tag{15}$$

$$\Delta T = T_{encrypt} - T_{raw} = (\hat{t}^c - t^c) + t^e \tag{16}$$

$t^e$ can be reduced by offloading the calculation of encryption module on more powerful computing nodes (e.g., switching from end devices to edge devices). $(\hat{t}^c - t^c)$ can be reduced by changing the output size of encryption module. In practice, given the computational capability of each device, it is also possible to find a best route for encryption with the least overhead. Due to the space limitation, we do not discuss this in this paper.

We further introduce the concept of dependable and undependable devices in our proposed framework. A dependable device is a computing node which is safe for user to upload the source data for calculation, while an undependable device is not safe. With the assumptions listed at the beginning of §3, all end devices and part of edge devices are treated as dependable devices, the rest of edge devices and the cloud are treated as undependable devices. Only dependable devices can receive the source data from users, and undependable devices can only receive encrypted data from the output of encryption module.

### 3.6 Fault Tolerance

To recover from device failure, we consider two methods, **Reassign** and **Monitor**, which gives fault tolerance to the system. In **Reassign**, after all tasks have been assigned at least once, unfinished tasks are reassigned to all devices regardless of their current state. This method is relatively easy to implement, but it lacks flexibility in dealing with device failures. Communication time overhead is generated whenever a task is assigned to a failed device, which triggers the timeout limit of the communication module. Specifically, with $N_t$ tasks sent to failed devices and the communication timeout $t_{timeout}$ the overhead of communication time, $OVT_{Reassign}$, is expressed in eq. 17. $OVT_{Reassign}$ becomes unacceptably large when many devices have failed.

$$OVT_{Reassign} = N_t \times t_{timeout} \tag{17}$$

To tackle the communication overhead, we propose another method, **Monitor**, which dynamically monitors the state of each device. Another process on the device responsible for sending tasks is started to query all available devices in a device list. Any device that does not respond to the query, or trigger a communication timeout is deleted from the device list. Only devices in the list can receive tasks, so that the number of communication timeout is significantly reduced.

The interval of queries is important for end devices due to their limited computation and network resources. With a small query interval, the list of available devices is updated quickly so that communication timeout is less likely to happen. However, the computation involved in computing and communicating may slow down other parts of the end devices. Thus a balanced query interval should be chosen according to real-world situations.

### 3.7 Communication Cost

The communication cost, or its quantization: the communication time $t^c$, for different devices to transfer data such as a batch of test images, encrypted data, and the result can be expressed, in ideal cases, as the sum of three parts: the cost for data transferred, the cost for communication protocol, and the cost for meta data. Suppose the bandwidth of a network link is $b$, then the communication time $t^c$ can be expressed, for simplicity, by dividing the total number of bytes $B$ to be transferred by the bandwidth.

$$t^c = (B_{data} + B_{proto} + B_{meta})/b \tag{18}$$

$$B_{data} \propto (B_{dtype} \times BS \times W \times H \times Ch) \tag{19}$$

where $B_{dtype}$ is the number of bytes for a specific data type, e.g., 4 bytes for a *float32* number; $BS$ stands for the batch size; $W$ (width), $H$ (height), and $Ch$ (channel) represent the dimension of data to be transferred; $B_{proto}$ is the additional bytes from the communication protocol, e.g., header information in TCP/IP protocol; $B_{meta}$ denotes the bytes of the meta data, i.e., the information of the current task. In our proposed framework, mainly three types of data are transferred among devices:

1) The source image provided by one of the end devices, consumed by any dependable device, where $W$ and $H$ are exactly the size of images, $Ch$ is 3 for RGB images (or 1 for grayscale images ), and $B_{dtype} = 1$ for regular *uint8* colored images.
2) The encrypted data generated by the encryption module , where $W$, $H$ and $Ch$ are the output size (i.e., width, height, and the number of filters) of the encryption convolutional layer in the module. $B_{dtype}$ is generally set to 4 bytes for *float32* numbers.
3) The inference result from computing nodes. $W = H = 1$, and $B_{dtype} = 1$ for *uint8* integers.

The cost of transferring the meta data , $B_{meta}$, mainly consists of three parts: the information of the source data (e.g., dimension, data type, with/without encryption, encryption device, etc.), the data transferring path for the current job, and the timestamp in each step (to evaluate the system performance).

## 4 System Evaluation

In this section, we evaluate our system on a setup with multiple types of distributed deices. We demonstrate various advantages of our framework, including but not limited to the following aspects:

1) Our framework works on heterogeneous distributed computing systems and provides better performance, lower response time, and better resource usage.
2) Our framework loads different neural networks on each device to balance among the performance, response time, and resource usage.

3) Our framework takes advantage of heterogeneous computation tasks to efficiently offload easier tasks to devices with less computational ability for better user experience.
4) Our framework utilizes mapping based scheduling algorithm for balanced response time and performance.
5) Our framework achieves privacy protection by using a single-layer neural network for encryption and separating dependable and undependable devices.
6) Our framework features fault tolerance by monitoring the state of each device and maintaining a list of available devices.

Combining the distributed computing node heterogeneity with the neural network heterogeneity, we demonstrate the actual workflow of our evaluation system as below.

1) Test images are first distributed to end devices for a quick, coarse-grained response.
2) Sensitive data is encrypted to protect the privacy of users if sending to undependable computing nodes, as discussed in §3.5.
3) According to the scheduling algorithm in §3.4, data is further sent to edge devices or the cloud for a slower, fine-grained response.

We first introduce the neural networks used in the our evaluation system in §4.1. Then the evaluation system architecture is presented in §4.2. §4.3 introduces the evaluation dataset, and §4.4 to §4.6 presents the impact of three types of heterogeneities. §4.7 to §4.9 presents our results regarding to scheduling algorithm and scalability, privacy protection, and fault tolerance.

### 4.1 Neural Networks In The Evaluation System

Instead of loading the same model on all types of devices [11], our framework introduces an approach to load different neural networks on different types of devices to better accommodate their features and restrictions. For end devices, which has very limited memory and computational resources, we adopt a tiny but powerful neural network, MobileNet [12]. Since the restriction of hardware is relatively loosen for edge devices, we adopt a much deeper, much stronger, and widely used DNN structure, Residual Network (ResNet) [3]. . The choice for cloud devices is all about performance, i.e., the accuracy of classification for our evaluation dataset. We choose a variant of Residual Network, the Wide Residual Network [46], which is both larger in model size and deeper than typical ResNet configurations. All models are trained off-line on the test server with the entire CIFAR100 training set. The detailed structures of aforementioned neural networks are shown in Figure 2.

We use MobileNet on end devices for its small memory size and fast inference speed. The network structure follows the original setup reported in the paper of MobileNet, with two global hyper-parameters: width multiplier $\alpha = 0.5$, and depth multiplier $\beta = 1$. The input size of MobileNet is changed to the dimension of images in CIFAR100 dataset, which is $[32 \times 32 \times 3]$, and the output size is changed to 20 for exactly 20 classes of coarse labels. The model is trained

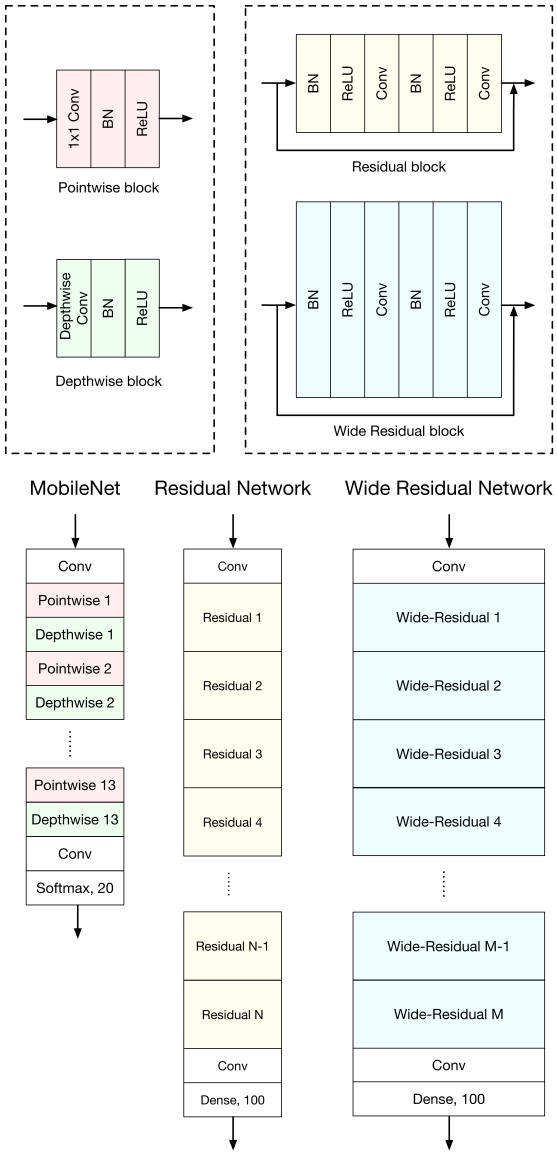


Fig. 2. Network structure of MobileNet [12], Residual Network (ResNet) [3], and Wide Residual Network (WRN) [46]. Red block and green block represents the Pointwise layer and Depthwise layer proposed in MobileNet, respectively. Yellow block represents the residual block proposed in ResNet, and blue block represents the wide residual block proposed in WRN. Best viewed in color.

with batch size of 32 for 100 epochs. We follow the training data augmentation methods reported in the original paper, and use Adam [54] as the optimizer with an initial learning rate of $0.01$.

On edge devices, we use ResNet with a 34-layer configuration to balance the speed and accuracy. The input size of ResNet is changed to fit the training images, and the output layer can produce classification for 100 labels, which is the number of fine-grained labels. The ResNet is trained with batch size of 32 for up to 500 epochs, and the model with the best performance (i.e., the lowest loss) is saved as the final model. The training data augmentation includes feature-wise mean subtraction, squeezing and randomly horizontally flip. We use Adam as the optimizer.

We adopt Wide Residual Network (WRN) on the cloud

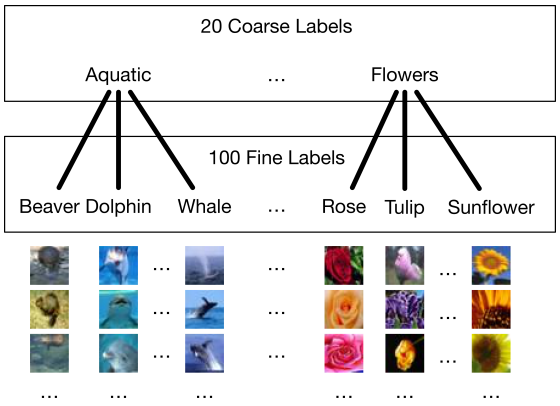


Fig. 3. Illustration of fine and coarse labels introduced in CIFAR100 [50].

for the best accuracy among all types of devices. Like ResNet, the WRN is trained for classification of 100 labels. We use the variant with depth parameter of 28, and width parameter of 10 in the model. It is trained with Stochastic Gradient Descent with Momentum of 0.9 and Nesterov method enabled. The learning rate starts from 0.1, and decays by a factor of 0.2 when the number of epochs reaches 60, 120, and 160. The training data is augmented by feature-wise centering, standard deviation normalization, and ZCA whitening. The best model trained for 200 epochs, with batch size of 128 is chosen as the final model for testing.

### 4.2 Evaluation System Architecture

The evaluation system architecture is illustrated in Figure 1. (e), which contains 8 end devices, 4 edge devices, and 1 cloud. We simulate the system using the combination of virtual machines (VMs), rather than a piece of program running locally on one machine as in DDNN, so that the communication between computing nodes are actually sent via network requests. We use 8 separate VMs each with one virtual CPU core running Ubuntu 16.04 as test machines for 8 end devices, 4 VMs each having 4 CPU cores for 4 edge devices and 1 terminal on the host machine with GPU acceleration enabled for the cloud. The host machine is equipped with dual Intel Xeon CPUs @2.4GHz and NVIDIA Tesla K80 graphics cards running on Ubuntu 16.04.

The communication between devices are implemented by HTTP requests, through a virtual network interface. In our simulation system, we manually set the process waiting time in each communication between two devices according to their types. Notice we only put restriction on the total communication time $t^c$ in the evaluation system to discuss its impact (more can be found in §3.7). In the following sections, if not specified, the simulated $t^c$ for the cloud is 1 second, and 0.5 seconds for edge devices.

### 4.3 Evaluation Dataset

We evaluate our system on CIFAR100 [50], which is a challenging image classification dataset containing 60K 32x32 colored natural images in total. There are originally 100 fine-grained labels in this dataset, representing 100 types of objects appeared in all images (e.g. apples, crab, and fox). 20 classes are proposed as the abstraction of those 100 classes, which are called superclasses, or coarse-grained classes. Some examples showing the relationship between coarse-grained classes and the fine-grained are illustrated in Figure 3. Images in each fine-grained class are separated into two parts, 500 images for training and 100 images for testing, which in total sums up to 50K training samples and 10K testing samples. Top-1 accuracy is chosen as the metric for evaluating the performance, which is the ratio of correct predictions among all test cases.

There are three main reasons why we choose this dataset in the evaluation system: (a). the image size is relatively small, which makes the simulation of computing on end devices possible, (b). the hierarchy structure of labels in this dataset meets our requirement of system task heterogeneity, and (c). the dataset is widely used for testing image classification algorithms.

In the following discussion, if it is not explicitly explained, we test our proposed framework for the classification job on the entire test set. The default batch size is 50, which means that each communication transfers 50 images sampled from the test set or 50 results correspondingly. Data pre-processing is implemented in the module of each neural network, and no global data pre-processing is done for test images. By default, we average the performance concerning accuracy or time through all tasks, more specifically 10K/50 = 200 tasks.The accuracy of coarse labels and fine labels are reported separately.

### 4.4 Impact of Distributed Computing Node Heterogeneity

To illustrate the impact of distributed computing node heterogeneity, we setup a baseline where the theoretical inference time, the classification accuracy for fine-grained and coarse-grained labels and the size of models are reported in Table 1. The inference time (prediction speed) in Table 1 is the theoretical upper bound for each type of neural network, on different devices. It is measured by continuously feeding the neural network with batched images without considering the overhead from communication functionality (different from the inference time in Table 2). The difference of time-stamp when each task starts and when it is finished is treated as the theoretical inference time for such neural network on a typical type of device for a single batch of images. It is then averaged over 200 runs, as explained at the end of §4.3. Without distributed computing node heterogeneity, mainstream solutions involve loading a powerful neural network on the cloud. The computational resources on edge and end devices are then wasted because they only take charge of generating source data or transferring those data. By distributing neural networks on end and edge devices, they can produce classification result together with the cloud. For example, suppose that we load MobileNet on end devices, ResNet on edge devices, and WRN on the cloud. Theoretical inference time for the three networks are relatively close to each other, with up to only 34% differences for the comparison between Cloud+WRN and End device+MobileNet. It shows that instead of uploading all data to the cloud for computing, making use of the distributed hierarchy is fully functional.

Another important factor with respect to the impact of computing node heterogeneity is the network latency.

TABLE 1
The comparison of the *theoretical* inference time, accuracy and model size for different neural networks on various types of computing nodes. The wall time to finish each task, divided by the batch size (we use 50 in practice) is referred to the inference time for each image (ms/image). The accuracy with coarse and fine labels, 20 and 100 labels respectively, are reported separately.

| Inference time (ms/image) | MobileNet | ResNet | WRN |
|---|---|---|---|
| Deploy on the Cloud | 0.25 | 0.79 | 6.99 |
| Deploy on Edge devices | 2.03 | 5.53 | 227 |
| Deploy on End devices | 5.20 | 15.7 | 1.17K |
| Testing accuracy | | | |
| Coarse labels | 0.50 | 0.73 | 0.83 |
| Fine labels | / | 0.61 | 0.73 |
| Model size | | | |
| Parameters (M) | 0.84 | 21.4 | 40.6 |
| Weights file (MB) | 10.3 | 85.7 | 324.9 |

TABLE 2
The inference time of neural networks on different types of devices, with various simulated communication latency in the distributed framework. 'Comm. latency' (short for communication latency) includes both the sending latency time and the receiving latency time. 'N/A' means the communication module is shut down, in the hope of comparing with zero latency to show the overhead of communication module.

| Device | Comm. latency (s) | Inference time (ms/image) |
|---|---|---|
| End device | N/A | 7.4 |
| | 0.0 | 10.4 |
| Edge device | N/A | 6.0 |
| | 0.0 | 8.4 |
| | 0.5 | 18.3 |
| Cloud | N/A | 7.4 |
| | 0.0 | 10.8 |
| | 1.0 | 31.0 |

We test the system inference speed on various devices with or without network latency, and with various latencies, in the actual distributed system, and the result is illustrated in Table 2. Notice, compared with Table 1, due to the computational overhead from other parts of the program, such as HTTP server, data packing and unpacking, and process synchronization, the actual inference time is slightly larger than theoretical inference time. With computing node heterogeneity, the system inference time has decreased a lot, from 31.0 (ms/image) to 10.4 (for end devices) or 18.3 (for edge devices). The advantage mainly comes from avoiding the long network latency that is probably unavoidable in traditional framework with only the cloud.

### 4.5 Impact of Neural Network Heterogeneity

One of the motivations for adopting neural network heterogeneity is to speed up the inference time on computing nodes with low computational resources, e.g., IoT devices. To demonstrate the impact of neural network heterogeneity, we also systematically compare the inference time, model size, and inference speed of each neural network on different types of devices, as illustrated in Table 1.

The model size is very important for end devices, especially with the rapidly growing depth of modern neural networks. Instead of loading giant neural networks on end devices, neural network heterogeneity allows us to put a compact neural networks with much fewer parameters and a much smaller weight file. As shown in the last two rows in Table 1, we can squeeze a neural network to an end device, which has up to 48x fewer parameters, and up to 32x smaller in terms of the size of weights file by comparing MobileNet with WRN. The benefit of smaller models on end devices is straightforward in terms of speed, where we achieve 3x and 225x less inference time compared with neural networks on edge devices and the cloud if they are tested on the same end device. On the other hand, if WRN is loaded on an end device, the inference time of 1.17(s) per image is going to block all other processes on the end device, making it totally unusable. The drawback in accuracy is less important for end devices, since one of the goals of distributing tasks on end devices is the low response time.

On the other hand, loading much deeper and stronger neural networks on edge devices or the cloud enables them to perform much better than end devices. Specifically, ResNet on edge devices is 46% better in terms of coarse label accuracy compared with MobileNet on end devices, and WRN on the cloud is 20% better regarding fine label accuracy compared with ResNet on edge devices. Since edge devices and the cloud do not have limitations on power consumption and moreover the cloud has no limited computational resources, they can deal with complex computation quickly. With more CPU cores enabled on edge devices (compared with end devices, where 4 vCPUs V.S. 1 vCPU in our simulation), edge devices can handle the inference of ResNet, which is 25x larger than MobileNet in terms of the number of parameters and with 2.8x less inference time than on end devices. The cloud, which is equipped with GPU produces even better speedup for larger and deeper neural networks. The inference time of WRN on edge devices is 32x more than that on the cloud, but the inference time of MobileNet on edge devices is only 8x more than that on the cloud, meaning that the cloud is even more suitable for a large scale DNN. In summary, with the help of neural network heterogeneity, all the three types of devices are better utilized according to their features, and the performance and response time for the system are enhanced.

### 4.6 Impact of System Task Heterogeneity

System task heterogeneity actually makes the neural networks on end devices realistically. It enhances user experience by generating a coarse response for less response time on a less powerful computing node. The accuracy of MobileNet for coarse labels is relatively usable, considering its fast inference speed. However, since the performance of coarse labels and fine labels cannot be compared head-

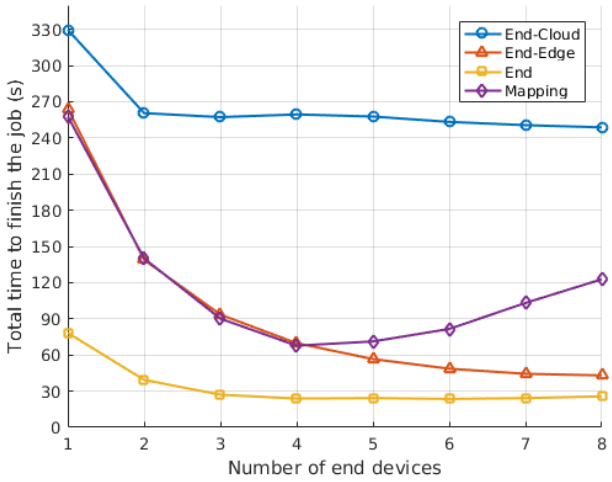


Fig. 4. The time to finish the classification task in different situations with various numbers of end devices. In all 4 schemes, end devices first produce a coarse response given the source image. In the End-Cloud and End-Edge scheme, the data is sent to the cloud and edge devices for further process. In the End-Edge Mapping scheme, the first 4 end devices send data to 4 edge devices correspondingly, and the rest of end devices send data to the cloud.

to-head, we emphasize the advantage with (and without) system task heterogeneity.

### 4.7 Evaluation for Scheduling and Scalability

Based on the discussion in §3.4, we test the impact of various scheduling algorithms and the scalability of proposed framework. The encryption module is enabled in this section for more realistic simulation. The detailed workflow of four scheduling algorithms are listed below.

1) *End*: Only process the data on end devices. All tasks are assigned sequentially through all possible end devices.
2) *End-Cloud*: Process the source image first on end devices for a coarse-grained result. Then encrypt and send to the cloud for a fine-grained result.
3) *End-Edge*: Process the source image first on end devices. Then send encrypted data to all edge devices (4 edge devices) for a fine-grained result.
4) *Mapping*: Use mapping scheduling scheme for end devices and edge devices. Send encrypted data from the rest of end devices to the cloud.

The comparison of the time to finish the task with four schemes ($T_{End}$, $T_{End-Cloud}$, $T_{End-Edge}$, and $T_{Mapping}$) with respect to the number of end devices is illustrated in Figure 4. It is observed that both *End-Cloud* and *End* situations have relatively poor scalability. In practice, as the number of end devices increases, the network delay between end devices and the cloud becomes the bottleneck for the *End-Cloud* scheme. All tasks are quickly processed on end devices, and encrypted data begins to stuck on the way to the cloud, waiting to be transferred. For the *End* scheme, the speed of assigning tasks becomes the bottleneck as the number of end devices becomes larger. The response time for each end device, for each task is faster than assigning $N_{end} - 1$ tasks, where $N_{end}$ is the number of end devices. The next time when an end device receives its task, the previous one is already finished. The speed of assigning tasks then becomes the bottleneck, instead of the processing speed on end devices or the communication delay.

TABLE 3
The testing accuracy with fine-grained labels, i.e. 100 classes, of *End-Edge* (EE), *Mapping* (Map.), and *End-Cloud* (EC) scheduling schemes with respect to the number of end devices.

| Accuracy | Number of end devices | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| EE | 0.61 | 0.61 | 0.61 | 0.61 | 0.61 | 0.61 | 0.61 | 0.61 |
| Map. | 0.61 | 0.61 | 0.61 | 0.61 | **0.63** | **0.65** | **0.66** | **0.67** |
| EC | 0.73 | 0.73 | 0.73 | 0.73 | 0.73 | 0.73 | 0.73 | 0.73 |

For the *End-Edge* and the *Mapping* scheme, the usage of edge device improves the scalability of distributed systems, compared with the *End* and *End-Cloud* scheme. There are mainly three reasons contributing to such advantages: (a). accessible multiple edge devices, which increase the throughput compared with one cloud device in *End-Cloud* scheme, (b). relatively lower network delay (0.5(s) for edge devices rather than 1(s) for the cloud), compared with *End-Cloud* scheme, and (c). relatively higher inference speed than end devices, which makes the total processing time become closer and closer to the *End* scheme with more and more end devices. For the *End-Edge Mapping* scheme, $N_{map}$ is set to 1 according to our evaluation system setup. The minor difference between the *Mapping* and the *End-Edge* scheme with 1-4 end devices in Figure 4 shows the efficiency of mapping end devices to edge devices according to their relative ratio of processing time. However, the communication time for the cloud restricts the scalability when more than 4 end devices are used in the *Mapping* scheme.

The comparison of overall accuracy for these scheduling schemes are listed in Table 3. The *End* scheme has the least preferred accuracy because of coarse-grained labels (0.50), while *End-Cloud* scheme has the highest accuracy. *End-Edge* scheme stays between them because the performance of neural networks on edge devices are naturally located between end devices and the cloud. The key feature of *Mapping* scheme is its changeable accuracy, according to the number of end devices. Its accuracy is exactly the same as *End-Edge* scheme when no more than 4 end devices are used, and gently becomes larger when more end devices are used.

TABLE 4
The specification of encryption modules for ResNet and WRN, including the output shape, the number of parameters, and the size of weights file. Encryption modules for ResNet and WRN are deployed on edge devices and the cloud respectively.

| | ResNet | WRN |
|---|---|---|
| Output shape (H x W x Ch) | $16 \times 16 \times 64$ | $32 \times 32 \times 16$ |
| Parameters | 9472 | 432 |
| Weights file (KB) | 50.1 | 11.9 |

With that said, further scheduling can be done with more information about the expected accuracy of the system.

### 4.8 Additional Cost for Privacy Protection

We use a single layer of convolutional layer stripped from the deeper neural network in the cloud or the edge device as the encryption module in the evaluation system. Table 4 illustrates the output size, the number of parameters, and the size of weight file of the encryption module for two types of DNNs on edge devices and the cloud. For comparison, the number of parameters in encryption modules for ResNet and WRN is only 1.1% and 0.05% of parameters in MobileNet, showing subtle computational overhead .

In the evaluation system, we have tested the overhead with and without encryption module, with different simulated values of network delay. Table 5 illustrates the overhead of processing time for each task (explained in §4.3) in two situations: (a). encrypt the data on an end device, then send to the cloud, and (b). encrypt the data on an edge device, then send to the cloud. It is observed that the difference of communication time $(\hat{t}^c - t^c)$ is subtle compared with the encryption time $t^e$. By encrypting on a stronger node, the overhead can be reduced from 13.1% (at 2(s) latency) to 29.8% (at 0.5(s) latency). Besides, the time overhead is less significant as the network latency goes higher (e.g., the overhead drops from 57.1% at 0.5(s) latency to 17.9% at 2(s) latency for end-cloud scheme). Since the network latency in real world is generally large and unstable, the experiment result suggests that the overhead in real system is considerably small. Further optimization on scheduling algorithm can be done for less overhead given the latency and computational capacity of each device.

### 4.9 Showcase of Fault Tolerance

Three methods with respect to the requirement of fault tolerance are tested in our evaluation system: (a). without any functionality dealing with device failure, (b). the **Reassign** method, and (c). the **Monitor** method. The duration from the beginning of assigning tasks to the time when all tasks are finished is recorded and averaged over five runs. We only consider end devices for simplicity, and we use all 8 end devices by default. We simulate device failures by letting them automatically shutdown after processing half the number of tasks that a worker is going to process, which is 12 in our case. The timeout of transferring data to another device, $t_{timeout}$, is set to 0.5 (s), which means at most 0.5 (s) delay is generated for each communication with end devices. Since the timeout is a simulated value, the interval of queries is set according to the timeout in the evaluation system. The curve of averaged duration time with respect to the number of device failure is illustrated in Figure 5, and the raw time data is listed in Table 6.

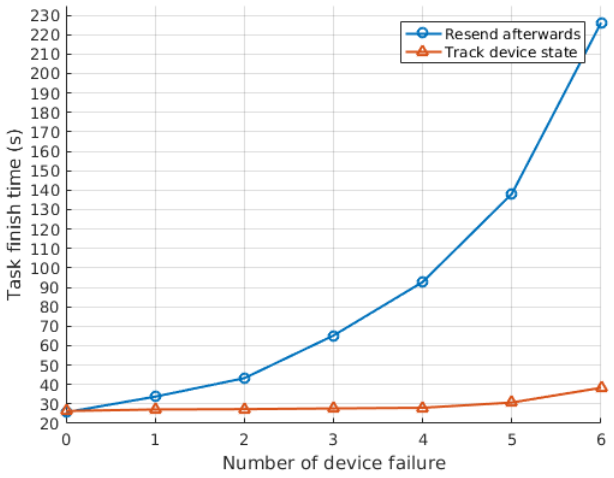


Fig. 5. The time to finish the classification task with different methods for fault tolerance, with different number of device failures.

Without any functionality dealing with device failure, the task can not be done if any device fails, which is labeled as $\infty$ in Table 6. With the functionality of reassigning sub-tasks enabled, the system is able to finish the entire task even if some devices fail, but for longer time than that with no device failure. Method two directly reassign any unfinished task to next end device in the original list, without considering and testing whether the device is available or not. The drawback is that for each communication to the failed device, the time for waiting the network timeout is wasted, causing an unacceptable increasing duration curve. With further consideration of fault tolerance in this system, we notice that each sub-task in this specific task is separable from each other, which means the order of assigning each sub-task is not important. To avoid the timeout overhead in method two, we have tested method three in our evaluation system, which has a separate sub-process periodically query each end device to check their availability. A global device list is maintained, containing devices which is available currently. When assigning a sub-task, the next device in the global device list is chosen as the worker. From Table 6, it is observed that this method generates considerably less over-

TABLE 5
The time overhead with different simulation values of network latency. The overhead is the percentage of additional time used with encryption module, compared with no encryption module used. In both cases, the encrypted/unencrypted is finally sent to the cloud for calculation.

| Speed overhead | Simulated network latency (s) | | | |
|---|---|---|---|---|
| | 0.5 | 1 | 1.5 | 2 |
| Encrypt on End devices | 57.1% | 33.2% | 24.7% | 17.9% |
| Encrypt on Edge devices | 44.0% | 27.2% | 20.6% | 15.9% |

TABLE 6
The time to finish the classification task with different fault recovering methods, with different number of end device failures.

| Finish time (s) | Number of end device failures | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| N/A | 24.9 | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| Reassign | 25.7 | 33.7 | 43.2 | 65.2 | 92.8 | 138 | 226 |
| Monitor | 26.4 | **27.2** | **27.3** | **27.7** | **28.0** | **30.7** | **38.3** |

head especially for large number of devices failures, e.g., 5.9x faster with 6 device failures compared with previous method. It is also observed that subtle overhead is generated by applying method two and three, 3.2% and 6.0% longer than method one respectively. However, considering their advantages of enabling fault tolerance for the system, the overhead is acceptable.

## 5 Conclusion and future work

In this paper, we propose a novel Heterogeneous Distributed Deep Neural Network (HDDNN) framework that is developed on hierarchical distributed systems, including the cloud, the edge (fog) devices and the end devices. We propose three types of heterogeneity based on a deeper observation on real-world distributed systems, with (a). distributed computing node heterogeneity, (b). neural network heterogeneity, and (c). system task heterogeneity. A simulated system of proposed framework is being tested on image classification jobs, showing that our framework is able to provide rich features on a highly heterogeneous distributed system. Illustrated by elaborately designed experiments, our proposed HDDNN framework offers low response time, optimized resource usage, high performance, better user experience, privacy protection and robustness.

Future works include (a). deploying our framework on heterogeneous hardwares, (b). testing for a real-world, task-specific dataset, (c). arguing and discussing about bottlenecks in the system under real-world situations, etc.

## Acknowledgments

This work was supported in part by National NSF of China (NO.61525204, 61732010, 61872234). Tao Song and Ruhui Ma are the Corresponding Authors.

## References

[1] G. Ghimpeteanu, T. Batard, M. Bertalmío, and S. Levine, "A decomposition framework for image denoising algorithms," *IEEE Trans. Image Processing*, vol. 25, no. 1, pp. 388–399, 2016.

[2] Y. Zhang and T. Funkhouser, "Deep depth completion of a single rgb-d image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[4] W. Wan, Y. Zhong, T. Li, and J. Chen, "Rethinking feature distribution for loss functions in image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[5] S. Ren, W. Chen, S. Liu, M. Li, M. Zhou, and S. Ma, "Triangular architecture for rare language translation," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018.

[6] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, M. Schuster, N. Shazeer, N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, Z. Chen, Y. Wu, and M. Hughes, "The best of both worlds: Combining recent advances in neural machine translation," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018.

[7] O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech & Language Processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[8] H. Seki, T. Hori, S. Watanabe, J. Le Roux, and J. R. Hershey, "A purely end-to-end system for multi-speaker speech recognition," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 2018.

[9] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, "Scalable distributed computing hierarchy: Cloud, fog and dew computing," *Open Journal of Cloud Computing*, vol. 2, no. 1, pp. 16–24, 2015.

[10] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "A scalable framework for wireless distributed computing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2643–2654, 2017.

[11] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *the 37th IEEE International Conference on Distributed Computing Systems*, 2017.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[13] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[14] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *J. Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[15] "Amazon aws," https://aws.amazon.com.

[16] "Microsoft azure," https://azure.microsoft.com/en-us/.

[17] J. Spillner, J. Müller, and A. Schill, "Creating optimal cloud storage systems," *Future Generation Comp. Syst.*, vol. 29, no. 4, pp. 1062–1072, 2013.

[18] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Tech. Rep., 2009.

[19] A. Botta, W. de Donato, V. Persico, and A. Pescapè, "Integration of cloud computing and internet of things: A survey," *Future Generation Comp. Syst.*, vol. 56, pp. 684–700, 2016.

[20] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, 2016.

[21] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[22] F. Bonomi, R. A. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile Cloud Computing*, 2012.

[23] A. G. Greenberg, J. R. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2009.

[24] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[25] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *The 12th Annual International Conference on Mobile Systems, Applications, and Services*, 2014.

[26] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the Sixth European conference on Computer systems*, 2011.

[27] C. Perera, A. B. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[28] A. Whitmore, A. Agarwal, and L. D. Xu, "The internet of things - A survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[29] A. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a service and big data," *arXiv preprint arXiv:1301.0159*, 2013.

[30] L. W. F. Chaves and C. Decker, "A survey on organic smart labels for the internet-of-things," in *Seventh International Conference on Networked Sensing Systems*, 2010.

[31] A. M.-H. Kuo, "Opportunities and challenges of cloud computing to improve health care services," *Journal of medical Internet research*, vol. 13, no. 3, 2011.

[32] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Trans. Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[33] M. S. Hossain and G. Muhammad, "Cloud-assisted industrial internet of things (iiot) - enabled framework for health monitoring," *Computer Networks*, vol. 101, pp. 192–202, 2016.
[34] M. M. Rathore, A. Ahmad, A. Paul, and S. Rho, "Urban planning and building smart cities based on the internet of things using big data analytics," *Computer Networks*, vol. 101, pp. 63–80, 2016.
[35] S. Chen, C. Lai, Y. Huang, and Y. Jeng, "Intelligent home-appliance recognition over iot cloud network," in *the 9th International Wireless Communications and Mobile Computing Conference*, 2013.
[36] F. Ding, A. Song, D. Zhang, E. Tong, Z. Pan, and X. You, "Interference-aware wireless networks for home monitoring and performance evaluation," *IEEE Transactions on Automation Science and Engineering*, vol. 99, pp. 1–12, 2017.
[37] F. Gao, "Vsaas model on dragon-lab," *International Journal of Multimedia & Ubiquitous Engineering*, vol. 8, no. 4, 2013.
[38] A. Prati, R. Vezzani, M. Fornaciari, and R. Cucchiara, "Intelligent video surveillance as a service," in *Intelligent multimedia surveillance*, 2013, pp. 1–16.
[39] C.-T. Fan, Y.-K. Wang, and C.-R. Huang, "Heterogeneous information fusion and visualization for a large-scale intelligent video surveillance system," *IEEE transactions on systems, man, and cybernetics: systems*, vol. 47, no. 4, pp. 593–604, 2017.
[40] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *arXiv preprint arXiv:1701.01090*, 2017.
[41] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Comp. Syst.*, vol. 78, pp. 680–698, 2018.
[42] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.
[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.
[44] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, pp. 211–252, 2015.
[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations*, 2015.
[46] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference 2016*, 2016.
[47] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems*, 2016.
[48] B. McDanel, S. Teerapittayanon, and H. T. Kung, "Embedded binarized neural networks," in *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, 2017.
[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[50] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Technical Report, University of Toronto*, 2009.
[51] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012.
[52] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "Firecaffe: Near-linear acceleration of deep neural network training on compute clusters," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
[53] P. H. Jin, Q. Yuan, F. N. Iandola, and K. Keutzer, "How to scale distributed deep learning?" in *Machine Learning Systems Workshop on Advances in Neural Information Processing Systems*, 2016.
[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

**Zongpu Zhang** is presently a senior student in the School of Electronic Information and Electrical Engineering at Shanghai Jiao Tong University. His research area includes machine learning application for image and video understanding and distributed computing.

**Tao Song** received his M.Eng. degree on Software Engineering major from Shanghai Jiao Tong University and B.Eng. degree on Automation from China University of Mining and Technology. He is currently a Ph.D. candidate in Shanghai Jiao Tong University in China. His research area includes data center networking, cloud computing and artificial intelligence.

**Liwei Lin** received the B.S. and M.S. degree from Fujian Normal University, China, in 2006 and 2010 repectively. He is currently pursuing the Ph.D. degree in computer science and engineering with Shanghai Jiao Tong University, China. His research interests include data center network, mobile computing, cloud computing and fog computing.

**Yang Hua** received the Ph.D. degree from Universit Grenoble Alpes/Inria Grenoble Rhne-Alpes, France, funded by the Microsoft Research Inria Joint Center. He is currently a Lecturer with the Queens University of Belfast, U.K. He holds three U.S. patents and one China patent. His research interests include machine learning methods for image and video understanding. He was a winner of the PASCAL Visual Object Classes Challenge Classification Competition in 2010, 2011, and 2012, and the Thermal Imagery Visual Object Tracking Competition in 2015.

**Xufeng He** received the Bachelor's degree in Microelectronics from Northwestern Polytechnical University, Xi'an, China in 2016. He is currently pursuing Msc in Computer Science from the Graduate School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University. His research interests include deep learning and distributed computing.

**Zhengui Xue** received her Ph.D. degree from the NUS Graduate School for Integrative Sciences and Engineering, National University of Singapore in 2013. She held a postdoctoral position at the Ecole Nationale des Travaux Publics de l'Etat from 2014 to 2015. She is currently an adjunct researcher in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. Her research interests include adaptive systems, knowledge-based intelligent control, optimal control, and machine learning.

**Ruhui Ma** received the Ph.D. degree in computer science from Shanghai Jiao Tong University (SJTU), China, in 2011. He held postdoctoral positions with SJTU (2012 and 2013) and McGill University, Canada (2014), respectively. He is currently an Associate Professor with the Department of Computer Science and Engineering, SJTU. His main research interests are in virtual machines, computer architecture, network virtualization and artificial intelligence.

**Haibing Guan** received his Ph.D. degree in computer science from the Tongji University (China), in 1999. He is currently a professor with the Faculty of Computer Science, Shanghai Jiao Tong University, Shanghai, China. He is the founder of Shanghai Key Laboratory of Scalable Computing and Systems. He also is a member of IEEE and CCF. His current research interests include, but are not limited to, computer architecture, parallel computing, compiling and virtualization.